# Parcheado del kernel para Xenomai

La nueva arquitectura del framework de tiempo real Xenomai 3 puede ejecutarse en paralelo con Linux como un sistema co-kernel, o de forma nativa sobre el kernel de Linux. En este último caso, el kernel se puede complementar con el parche PREEMPT-RT para cumplir con requisitos de tiempo de respuesta más estrictos que el kernel estándar. La versión dual del kernel recibe el nombre de Cobalt, mientras que la versión que corre sobre el kernel Linux nativo se denomina Mercury.

En esta práctica vamos a aplicar los parches necesarios para instalar el core Cobalt de la infraestructura Xenomai 3 sobre el kernel 5.10 que viene con el raspberry pi OS.

## 1 Interrupt pipeline

El core Cobalt necesita que el kernel original sea parcheado para introducir una capa de virtualización de interrupciones. Esta capa permite al co-kernel tratar las interrupciones con mayor prioridad que Linux, delegando en éste las interrupciones que no trate. El kernel Linux maneja un controlador virtual de interrupciones, de forma que si quiere deshabilitarlas sólo lo hace para Linux, pero el co-kernel Cobalt sigue pudiendo tratar las interrupciones. De esta forma, el kernel Linux no puede retrasar el tratamiento de las interrupciones por el co-kernel.

Según la versión del kernel, la arquitectura de este pipeline es:

- Dovetail, para versiones de kernel a partir de la 5.10
- I-pipe, para versiones del kernel desde la 4.14 hasta la 5.9
- I-pipe legacy, para versiones del kernel anteriores

En nuestro caso tenemos un kernel 5.10, tenemos que usar Dovetail.

## 2 Parche para Dovetail

La información oficial para la instalación de estos parches se mantiene en una wiki accesible en https://source.denx.de/Xenomai/xenomai/-/wikis/home. El

mantenimiento de esta wiki tiene algunos defectos, y en la fecha de redacción de este documento el procedimiento de instalación no está actualizado para dovetail.

Aunque el proyecto xenomai viene con un script que aplica el parche dovetail, dicho parche sólo aplica sin problemas sobre un kernel vanilla. Sin embargo, a nosotros nos interesa parchear el kernel de raspberrypi, con los parches incluídos por ellos. Por este motivo vamos a aplicar primero el parche y tendremos que resolver manualmente los conflictos que surjan. Para ello lo primero que haremos será descargarnos el parche de dovetail más apropiado para nuestro kernel de https://xenomai.org/downloads/dovetail/. Igual que hicimos con PREEMPT_RT para encontrar la versión que necesitamos ejecutamos:

```
$ uname -r
5.10.92-v7+
```

Buscamos entonces la versión más proxima. Encontramos que hay una versión del parche dovetail para la 5.10.89, descargamos esa. Nos descargamos esta versión en el mismo directorio de trabajo que utilizamos para trabajar con el parche PREEMPT_RT:

```
$ cd ~/rpi-kernel
$ wget https://xenomai.org/downloads/dovetail/patch-5.10.89-dovetail1.patch.bz2
```

Una vez tenemos el parche descargamos la versión del kernel apropiada. Como hicimos con PREEMPT_RT consultaremos la web del kernel https://github.com/raspberrypi/linux, seleccionaremos el branch rpi-5.10.y y el fichero Makefile. Pincharemos en History y buscaremos la versión que coincida con la versión del kernel instalada.

En el momento de preparar este documento, con el sistema actualizado (sudo apt full-upgrade), la versión del kernel es 5.10.92-v7+. Por tanto bucamos en History la versión 5.10.92. Vemos que la entrada del 16 de enero está marcada con 5.10.92. Ésta es una versión upstream del kernel, sin las modificaciones para la raspberry pi, copiamos el sha de la del 17 de Enero (650082a559a570d6c9d2739ecc62843d6f951059), que tiene el merge con el remote tracking branch 'stable/linux-5.10.y'.

```
$ mkdir linux-xenomai
$ cd linux-xenomai
$ git init
$ git remote add origin https://github.com/raspberry/linux
$ git fetch --depth 1 origin 650082a559a570d6c9d2739ecc62843d6f951059
$ git checkout -b 5.10.92 FETCH_HEAD
```

Ahora podemos aplicar el parche de xenomai:

```
$ bzcat ../patch-5.10.89-dovetail1.patch.bz2 | patch -p1 | grep FAILED
Hunk #7 FAILED at 177.
1 out of 7 hunks FAILED -- saving rejects to file arch/arm/include/asm/irqflags.h.rej
Hunk #1 FAILED at 92.
```

```
1 out of 1 hunk FAILED -- saving rejects to file arch/arm/kernel/Makefile.rej
Hunk #3 FAILED at 458.
Hunk #6 FAILED at 489.
Hunk #15 FAILED at 1042.
3 out of 15 hunks FAILED -- saving rejects to file drivers/dma/bcm2835-dma.c.rej
Hunk #1 FAILED at 102.
1 out of 1 hunk FAILED -- saving rejects to file drivers/irqchip/irq-bcm2835.c.rej
Hunk #1 FAILED at 1079.
1 out of 4 hunks FAILED -- saving rejects to file drivers/spi/spi-bcm2835.c.rej
```

Como vemos el parche no aplica limpiamente, ya que no era la versión original del kernel 5.10.89, sino una 5.10.92 parcheada por raspberry pi. Debemos resolver manualmente estos fallos. Procederemos uno a uno, abriendo en una ventana el fichero .rej con los cambios no aplicados, en otra ventana el fichero a parchear y en otra ventana podemos tener el fichero original (.orig). Puede ser util también descargar la versión del kernel linux 5.10.89 sin parchear para ver sobre qué código se diseñó el parche.

## 2.1 Fichero arch/arm/include/asm/irqflags.h

Comenzamos por ejemplo con arch/arm/include/asm/irqflags.h. El fichero .rej tiene el siguiente contenido:

```
--- arch/arm/include/asm/irqflags.h
+++ arch/arm/include/asm/irqflags.h
@@ -177,21 +188,28 @@ static inline unsigned long arch_local_save_flags(void)
  * restore saved IRQ & FIQ state
  */
 #define arch_local_irq_restore arch_local_irq_restore
-static inline void arch_local_irq_restore(unsigned long flags)
+static inline void native_irq_restore(unsigned long flags)
 {
    asm volatile(
-       "   msr " IRQMASK_REG_NAME_W ", %0  @ local_irq_restore"
+       "   msr " IRQMASK_REG_NAME_W ", %0  @ native_irq_restore"
        :
        : "r" (flags)
        : "memory", "cc");
 }

 #define arch_irqs_disabled_flags arch_irqs_disabled_flags
-static inline int arch_irqs_disabled_flags(unsigned long flags)
+static inline int native_irqs_disabled_flags(unsigned long flags)
 {
    return flags & IRQMASK_I_BIT;
 }
```

```
+static inline bool native_irqs_disabled(void)
+{
+   unsigned long flags = native_save_flags();
+   return native_irqs_disabled_flags(flags);
+}
+
+#include <asm/irq_pipeline.h>
 #include <asm-generic/irqflags.h>

 #endif /* ifdef __KERNEL__ */
```

Entonces abrimos el fichero arch/arm/include/asm/irqflags.h y buscamos la definición de la función arch_local_irq_restore:

```
define arch_local_irq_restore arch_local_irq_restore
static inline void arch_local_irq_restore(unsigned long flags)
{
    unsigned long temp = 0;
    flags &= ~(1 << 6);
    asm volatile (
        " mrs %0, cpsr"
        : "=r" (temp)
        :
        : "memory", "cc");
        /* Preserve FIQ bit */
        temp &= (1 << 6);
        flags = flags | temp;
    asm volatile (
        "    msr    cpsr_c, %0    @ local_irq_restore"
        :
        : "r" (flags)
        : "memory", "cc");
}
```

Como vemos la función es diferente a la que originalmente se quería parchear, pero en este caso el cambio es fácil de aplicar:

```
define arch_local_irq_restore arch_local_irq_restore
static inline void native_irq_restore(unsigned long flags)
{
    unsigned long temp = 0;
    flags &= ~(1 << 6);
    asm volatile (
        " mrs %0, cpsr"
        : "=r" (temp)
        :
        : "memory", "cc");
        /* Preserve FIQ bit */
```

```
        temp &= (1 << 6);
        flags = flags | temp;
    asm volatile (
        "    msr    cpsr_c, %0    @ native_irq_restore"
        :
        : "r" (flags)
        : "memory", "cc");
}
```

El cambio en la función arch_irqs_disabled_flags es también sencillo:

```
static inline int native_irqs_disabled_flags(unsigned long flags)
 {
    return flags & IRQMASK_I_BIT;
 }
```

Y finalmente hay que añadir la declaración de una nueva función e incluir un fichero de cabecera:

```
static inline bool native_irqs_disabled(void)
{
    unsigned long flags = native_save_flags();
    return native_irqs_disabled_flags(flags);
}

include <asm/irq_pipeline.h>
```

Una vez completado este proceso podemos borrar el fichero .rej y el fichero .orig correspondiente:

```
$ rm arch/arm/include/asm/irqflags.h.rej arch/arm/include/asm/irqflags.h.orig
```

## 2.2 Fichero arch/arm/kernel/Makefile

El parche rechazado (.rej) del siguiente fichero es:

```
--- arch/arm/kernel/Makefile
+++ arch/arm/kernel/Makefile
@@ -92,6 +92,11 @@ obj-$(CONFIG_PARAVIRT)   += paravirt.o
 head-y         := head$(MMUEXT).o
 obj-$(CONFIG_DEBUG_LL) += debug.o
 obj-$(CONFIG_EARLY_PRINTK) += early_printk.o
+ifeq ($(CONFIG_DEBUG_LL),y)
+obj-$(CONFIG_RAW_PRINTK)   += raw_printk.o
+endif
+
+obj-$(CONFIG_IRQ_PIPELINE) += irq_pipeline.o

 # This is executed very early using a temporary stack when no memory allocator
```

```
 # nor global data is available. Everything has to be allocated on the stack.
```

Abrimos el fichero Makefile y buscamos la línea que incluye early_printk.o:

```
obj-$(CONFIG_DEBUG_LL)  += debug.o
obj-$(CONFIG_EARLY_PRINTK)  += early_printk.o
obj-$(CONFIG_ARM_PATCH_PHYS_VIRT)   += phys2virt.o
```

En este caso el parche es muy sencillo de aplicar, basta con añadir las líneas que faltan:

```
obj-$(CONFIG_DEBUG_LL)  += debug.o
obj-$(CONFIG_EARLY_PRINTK)  += early_printk.o
ifeq ($(CONFIG_DEBUG_LL),y)
obj-$(CONFIG_RAW_PRINTK)    += raw_printk.o
endif

obj-$(CONFIG_IRQ_PIPELINE)  += irq_pipeline.o
obj-$(CONFIG_ARM_PATCH_PHYS_VIRT)   += phys2virt.o
```

Como hicimos con el fichero anterior, borramos el fichero .rej y el correspondiente .orig:

```
$ rm arch/arm/kernel/Makefile.rej arch/arm/kernel/Makefile.orig
```

## 2.3 Fichero drivers/dma/bcm2835-dma.c

El parche rechazado para el siguiente fichero es más largo:

```
--- drivers/dma/bcm2835-dma.c
+++ drivers/dma/bcm2835-dma.c
@@ -458,10 +469,41 @@ static void bcm2835_dma_start_desc(struct bcm2835_chan *c)

     list_del(&vd->node);

-    c->desc = d = to_bcm2835_dma_desc(&vd->tx);
+    c->desc = to_bcm2835_dma_desc(&vd->tx);
+    if (!bcm2835_dma_oob_capable() || !vchan_oob_pulsed(vd))
+        bcm2835_dma_enable_channel(c);
+}

-    writel(d->cb_list[0].paddr, c->chan_base + BCM2835_DMA_ADDR);
-    writel(BCM2835_DMA_ACTIVE, c->chan_base + BCM2835_DMA_CS);
+static bool do_channel(struct bcm2835_chan *c, struct bcm2835_desc *d)
+{
+    struct dmaengine_desc_callback cb;
+
+    if (running_oob()) {
+        if (!vchan_oob_handled(&d->vd))
```

```
+           return false;
+       dmaengine_desc_get_callback(&d->vd.tx, &cb);
+       if (dmaengine_desc_callback_valid(&cb)) {
+           vchan_unlock(&c->vc);
+           dmaengine_desc_callback_invoke(&cb, NULL);
+           vchan_lock(&c->vc);
+       }
+       return true;
+   }
+
+   if (d->cyclic) {
+       /* call the cyclic callback */
+       vchan_cyclic_callback(&d->vd);
+   } else if (!readl(c->chan_base + BCM2835_DMA_ADDR)) {
+       vchan_cookie_complete(&c->desc->vd);
+       bcm2835_dma_start_desc(c);
+   }
+
+   return true;
+}
+
+static inline bool is_base_irq_handler(void)
+{
+   return !bcm2835_dma_oob_capable() || running_oob();
 }

 static irqreturn_t bcm2835_dma_callback(int irq, void *data)
@@ -489,22 +532,27 @@ static irqreturn_t bcm2835_dma_callback(int irq, void *data)
     * if this IRQ handler is threaded.) If the channel is finished, it
     * will remain idle despite the ACTIVE flag being set.
     */
-   writel(BCM2835_DMA_INT | BCM2835_DMA_ACTIVE,
-          c->chan_base + BCM2835_DMA_CS);
+   if (is_base_irq_handler())
+       writel(BCM2835_DMA_INT | BCM2835_DMA_ACTIVE,
+           c->chan_base + BCM2835_DMA_CS);

    d = c->desc;
+   if (!d)
+       goto out;

-   if (d) {
-       if (d->cyclic) {
-           /* call the cyclic callback */
-           vchan_cyclic_callback(&d->vd);
-       } else if (!readl(c->chan_base + BCM2835_DMA_ADDR)) {
```

```
-           vchan_cookie_complete(&c->desc->vd);
-           bcm2835_dma_start_desc(c);
-       }
+   if (bcm2835_dma_oob_capable() && running_oob()) {
+       /*
+        * If we cannot process this from the out-of-band
+        * stage, schedule a callback from in-band context.
+        */
+       if (!do_channel(c, d))
+           irq_post_inband(irq);
+   } else {
+       do_channel(c, d);
    }

-   spin_unlock_irqrestore(&c->vc.lock, flags);
+out:
+   vchan_unlock_irqrestore(&c->vc, flags);

    return IRQ_HANDLED;
 }
@@ -1042,10 +1138,10 @@ static int bcm2835_dma_probe(struct platform_device *pdev)
            continue;

        /* check if there are other channels that also use this irq */
-       irq_flags = 0;
+       irq_flags = IS_ENABLED(CONFIG_DMA_BCM2835_OOB) ? IRQF_OOB : 0;
        for (j = 0; j <= BCM2835_DMA_MAX_DMA_CHAN_SUPPORTED; j++)
            if ((i != j) && (irq[j] == irq[i])) {
-               irq_flags = IRQF_SHARED;
+               irq_flags |= IRQF_SHARED;
                break;
            }
```

Abrimos entonces el fichero correspondiente y buscamos la definición de la función bcm2835_dma_start_desc:

```
static void bcm2835_dma_start_desc(struct bcm2835_chan *c)
{
    struct virt_dma_desc *vd = vchan_next_desc(&c->vc);

    if (!vd) {
        c->desc = NULL;
        return;
    }

    list_del(&vd->node);
```

```
    c->desc = d = to_bcm2835_dma_desc(&vd->tx);

    if (c->is_40bit_channel) {
        writel(to_bcm2711_cbaddr(d->cb_list[0].paddr),
               c->chan_base + BCM2711_DMA40_CB);
        writel(BCM2711_DMA40_ACTIVE | BCM2711_DMA40_CS_FLAGS(c->dreq),
               c->chan_base + BCM2711_DMA40_CS);
    } else {
        writel(d->cb_list[0].paddr, c->chan_base + BCM2835_DMA_ADDR);
        writel(BCM2835_DMA_ACTIVE | BCM2835_DMA_CS_FLAGS(c->dreq),
               c->chan_base + BCM2835_DMA_CS);
    }
}
```

Este parche es un poco más difíl de aplicar, si uno descarga la versión original del kernel, en ella esta función es:

```
static void bcm2835_dma_start_desc(struct bcm2835_chan *c)
{
    struct virt_dma_desc *vd = vchan_next_desc(&c->vc);
    struct bcm2835_desc *d;

    if (!vd) {
        c->desc = NULL;
        return;
    }

    list_del(&vd->node);

    c->desc = d = to_bcm2835_dma_desc(&vd->tx);

    writel(d->cb_list[0].paddr, c->chan_base + BCM2835_DMA_ADDR);
    writel(BCM2835_DMA_ACTIVE, c->chan_base + BCM2835_DMA_CS);
}
```

Sobre esta función es fácil entender lo que quería hacer el parche, quería transformar la función en:

```
static void bcm2835_dma_start_desc(struct bcm2835_chan *c)
{
    struct virt_dma_desc *vd = vchan_next_desc(&c->vc);

    if (!vd) {
        c->desc = NULL;
        return;
    }

    list_del(&vd->node);
```

```
    c->desc = to_bcm2835_dma_desc(&vd->tx);
    if (!bcm2835_dma_oob_capable() || !vchan_oob_pulsed(vd))
        bcm2835_dma_enable_channel(c);
}
```

Si buscamos en el fichero parcheado la función bcm2835_dma_enable_channel encontramos lo siguiente:

```
static inline void bcm2835_dma_enable_channel(struct bcm2835_chan *c)
{
    writel(c->desc->cb_list[0].paddr, c->chan_base + BCM2835_DMA_ADDR);
    writel(BCM2835_DMA_ACTIVE, c->chan_base + BCM2835_DMA_CS);
}
```

Como vemos son las líneas que se eliminan de la función bcm2835_dma_start_desc.

Sin embargo, el parche de raspberry pi dejó la función bcm2835_dma_start_desc como está en el correspondiente fichero .orig:

```
static void bcm2835_dma_start_desc(struct bcm2835_chan *c)
{
    struct virt_dma_desc *vd = vchan_next_desc(&c->vc);
    struct bcm2835_desc *d;

    if (!vd) {
        c->desc = NULL;
        return;
    }

    list_del(&vd->node);

    c->desc = d = to_bcm2835_dma_desc(&vd->tx);

    if (c->is_40bit_channel) {
        writel(to_bcm2711_cbaddr(d->cb_list[0].paddr),
               c->chan_base + BCM2711_DMA40_CB);
        writel(BCM2711_DMA40_ACTIVE | BCM2711_DMA40_CS_FLAGS(c->dreq),
               c->chan_base + BCM2711_DMA40_CS);
    } else {
        writel(d->cb_list[0].paddr, c->chan_base + BCM2835_DMA_ADDR);
        writel(BCM2835_DMA_ACTIVE | BCM2835_DMA_CS_FLAGS(c->dreq),
               c->chan_base + BCM2835_DMA_CS);
    }
}
```

Está claro entonces que para aplicar el primer trozo fallido del parche debemos modificar tanto la función bcm2835_dma_start_desc como la función bcm2835_dma_enable_channel:

```
static inline void bcm2835_dma_enable_channel(struct bcm2835_chan *c)
{
    if (c->is_40bit_channel) {
        writel(to_bcm2711_cbaddr(c->desc->cb_list[0].paddr),
               c->chan_base + BCM2711_DMA40_CB);
        writel(BCM2711_DMA40_ACTIVE | BCM2711_DMA40_CS_FLAGS(c->dreq),
               c->chan_base + BCM2711_DMA40_CS);
    } else {
        writel(c->desc->cb_list[0].paddr, c->chan_base + BCM2835_DMA_ADDR);
        writel(BCM2835_DMA_ACTIVE | BCM2835_DMA_CS_FLAGS(c->dreq),
               c->chan_base + BCM2835_DMA_CS);
    }
}

static void bcm2835_dma_start_desc(struct bcm2835_chan *c)
{
    struct virt_dma_desc *vd = vchan_next_desc(&c->vc);

    if (!vd) {
        c->desc = NULL;
        return;
    }

    list_del(&vd->node);

    c->desc = to_bcm2835_dma_desc(&vd->tx);
    if (!bcm2835_dma_oob_capable() || !vchan_oob_pulsed(vd))
        bcm2835_dma_enable_channel(c);
}
```

El resto del parche es más fácil de aplicar. Debajo de la función se añade la definición de otras dos funciones:

```
static bool do_channel(struct bcm2835_chan *c, struct bcm2835_desc *d)
{
    struct dmaengine_desc_callback cb;

    if (running_oob()) {
        if (!vchan_oob_handled(&d->vd))
            return false;
        dmaengine_desc_get_callback(&d->vd.tx, &cb);
        if (dmaengine_desc_callback_valid(&cb)) {
            vchan_unlock(&c->vc);
            dmaengine_desc_callback_invoke(&cb, NULL);
            vchan_lock(&c->vc);
        }
        return true;
```

```
    }

    if (d->cyclic) {
        /* call the cyclic callback */
        vchan_cyclic_callback(&d->vd);
    } else if (!readl(c->chan_base + BCM2835_DMA_ADDR)) {
        vchan_cookie_complete(&c->desc->vd);
        bcm2835_dma_start_desc(c);
    }

    return true;
}

static inline bool is_base_irq_handler(void)
{
    return !bcm2835_dma_oob_capable() || running_oob();
}
```

Luego buscamos la definición de la función bcm2835_dma_callback y en ella el comentario que incluye "despite the ACTIVE". En la versión original del kernel este fragmento era:

```
    /*
     * Clear the INT flag to receive further interrupts. Keep the channel
     * active in case the descriptor is cyclic or in case the client has
     * already terminated the descriptor and issued a new one. (May happen
     * if this IRQ handler is threaded.) If the channel is finished, it
     * will remain idle despite the ACTIVE flag being set.
     */
    writel(BCM2835_DMA_INT | BCM2835_DMA_ACTIVE,
           c->chan_base + BCM2835_DMA_CS);

    d = c->desc;

    if (d) {
        if (d->cyclic) {
            /* call the cyclic callback */
            vchan_cyclic_callback(&d->vd);
        } else if (!readl(c->chan_base + BCM2835_DMA_ADDR)) {
            vchan_cookie_complete(&c->desc->vd);
            bcm2835_dma_start_desc(c);
        }
    }

    spin_unlock_irqrestore(&c->vc.lock, flags);

    return IRQ_HANDLED;
```

```
}
```

Podemos ver que el parche pretende transformar este código en:

```
    /*
     * Clear the INT flag to receive further interrupts. Keep the channel
     * active in case the descriptor is cyclic or in case the client has
     * already terminated the descriptor and issued a new one. (May happen
     * if this IRQ handler is threaded.) If the channel is finished, it
     * will remain idle despite the ACTIVE flag being set.
     */
    if (is_base_irq_handler())
        writel(BCM2835_DMA_INT | BCM2835_DMA_ACTIVE,
            c->chan_base + BCM2835_DMA_CS);

    d = c->desc;
    if (!d)
        goto out;

    if (bcm2835_dma_oob_capable() && running_oob()) {
        /*
         * If we cannot process this from the out-of-band
         * stage, schedule a callback from in-band context.
         */
        if (!do_channel(c, d))
            irq_post_inband(irq);
    } else {
        do_channel(c, d);
    }

out:
    vchan_unlock_irqrestore(&c->vc, flags);

    return IRQ_HANDLED;
}
```

Sin embargo, nuestro código original, con los cambios introducidos para la raspberry pi, es:

```
    /*
     * Clear the INT flag to receive further interrupts. Keep the channel
     * active in case the descriptor is cyclic or in case the client has
     * already terminated the descriptor and issued a new one. (May happen
     * if this IRQ handler is threaded.) If the channel is finished, it
     * will remain idle despite the ACTIVE flag being set.
     */
    writel(BCM2835_DMA_INT | BCM2835_DMA_ACTIVE | BCM2835_DMA_CS_FLAGS(c->dreq),
           c->chan_base + BCM2835_DMA_CS);
```

```
    d = c->desc;

    if (d) {
        if (d->cyclic) {
            /* call the cyclic callback */
            vchan_cyclic_callback(&d->vd);
        } else if (!readl(c->chan_base + BCM2835_DMA_ADDR)) {
            vchan_cookie_complete(&c->desc->vd);
            bcm2835_dma_start_desc(c);
        }
    }

    spin_unlock_irqrestore(&c->vc.lock, flags);

    return IRQ_HANDLED;
}
```

En este caso es fácil deducir que la versión correctamente parcheada sería:

```
    /*
     * Clear the INT flag to receive further interrupts. Keep the channel
     * active in case the descriptor is cyclic or in case the client has
     * already terminated the descriptor and issued a new one. (May happen
     * if this IRQ handler is threaded.) If the channel is finished, it
     * will remain idle despite the ACTIVE flag being set.
     */
    if (is_base_irq_handler())
        writel(BCM2835_DMA_INT | BCM2835_DMA_ACTIVE |
            BCM2835_DMA_CS_FLAGS(c->dreq), c->chan_base + BCM2835_DMA_CS);

    d = c->desc;
    if (!d)
        goto out;

    if (bcm2835_dma_oob_capable() && running_oob()) {
        /*
         * If we cannot process this from the out-of-band
         * stage, schedule a callback from in-band context.
         */
        if (!do_channel(c, d))
            irq_post_inband(irq);
    } else {
        do_channel(c, d);
    }

out:
```

```
    vchan_unlock_irqrestore(&c->vc, flags);

    return IRQ_HANDLED;
}
```

Finalmente, buscamos la función bcm2835_dma_probe y en ella el comentario "check if there are other channels that also use this irq":

```
        /* check if there are other channels that also use this irq */
        /* FIXME: This will fail if interrupts are shared across
           instances */
        irq_flags = 0;
        for (j = 0; j <= BCM2835_DMA_MAX_DMA_CHAN_SUPPORTED; j++)
            if ((i != j) && (irq[j] == irq[i])) {
                irq_flags = IRQF_SHARED;
                break;
            }
```

Vemos que es fácil aplicar el último trozo del parche, que dejaría la función en:

```
        /* check if there are other channels that also use this irq */
        /* FIXME: This will fail if interrupts are shared across
           instances */
        irq_flags = IS_ENABLED(CONFIG_DMA_BCM2835_OOB) ? IRQF_OOB : 0;
        for (j = 0; j <= BCM2835_DMA_MAX_DMA_CHAN_SUPPORTED; j++)
            if ((i != j) && (irq[j] == irq[i])) {
                irq_flags |= IRQF_SHARED;
                break;
            }
```

Como antes, podemos borrar los ficheros .rej y .orig:

```
$ rm drivers/dma/bcm2835-dma.c.rej drivers/dma/bcm2835-dma.c.orig
```

## 2.4 Fichero drivers/irqchip/irq-bcm2835.c

El fragmento rechazado de este parche es:

```
--- drivers/irqchip/irq-bcm2835.c
+++ drivers/irqchip/irq-bcm2835.c
@@ -102,7 +102,8 @@ static void armctrl_unmask_irq(struct irq_data *d)
 static struct irq_chip armctrl_chip = {
     .name = "ARMCTRL-level",
     .irq_mask = armctrl_mask_irq,
-    .irq_unmask = armctrl_unmask_irq
+    .irq_unmask = armctrl_unmask_irq,
+    .flags = IRQCHIP_PIPELINE_SAFE,
 };
```

```
 static int armctrl_xlate(struct irq_domain *d, struct device_node *ctrlr,
```

Si buscamos la declaración de la variable armctrl_chip en el fichero irq-bcm2835.c encontramos lo siguiente:

```
static struct irq_chip armctrl_chip = {
    .name = "ARMCTRL-level",
    .irq_mask = armctrl_mask_irq,
    .irq_unmask = armctrl_unmask_irq,
ifdef CONFIG_ARM64
    .irq_ack    = armctrl_ack_irq
endif
};
```

La aplicación correcta del parche dejaría este fragmento de código como sigue:

```
static struct irq_chip armctrl_chip = {
    .name = "ARMCTRL-level",
    .irq_mask = armctrl_mask_irq,
    .irq_unmask = armctrl_unmask_irq,
    .flags = IRQCHIP_PIPELINE_SAFE,
ifdef CONFIG_ARM64
    .irq_ack    = armctrl_ack_irq
endif
};
```

Después podemos borrar los ficheros .rej y .orig correspondientes:

```
$ rm drivers/irqchip/irq-bcm2835.c.rej drivers/irqchip/irq-bcm2835.c.orig
```

## 2.5 Fichero drivers/spi/spi-bcm2835.c

El último fragmento de parche que no se aplicó limpiamente corresponde al fichero spi-bcm2835.c:

```
--- drivers/spi/spi-bcm2835.c
+++ drivers/spi/spi-bcm2835.c
@@ -1079,17 +1079,10 @@ static int bcm2835_spi_transfer_one_poll(struct spi_controller *ctl
    return 0;
 }

-static int bcm2835_spi_transfer_one(struct spi_controller *ctlr,
-                   struct spi_device *spi,
-                   struct spi_transfer *tfr)
+static unsigned long bcm2835_get_clkdiv(struct bcm2835_spi *bs, u32 spi_hz,
+                   u32 *effective_speed_hz)
 {
-   struct bcm2835_spi *bs = spi_controller_get_devdata(ctlr);
```

```
-   unsigned long spi_hz, cdiv;
-   unsigned long hz_per_byte, byte_limit;
-   u32 cs = bs->prepare_cs[spi->chip_select];
-
-   /* set clock */
-   spi_hz = tfr->speed_hz;
+   unsigned long cdiv;

    if (spi_hz >= bs->clk_hz / 2) {
        cdiv = 2; /* clk_hz/2 is the fastest we can go */
```

Para aplicarlo buscamos la función bcm2835_spi_transfer_one en el fichero:

```
static int bcm2835_spi_transfer_one(struct spi_controller *ctlr,
                    struct spi_device *spi,
                    struct spi_transfer *tfr)
{
    struct bcm2835_spi *bs = spi_controller_get_devdata(ctlr);
    unsigned long spi_hz, cdiv;
    unsigned long hz_per_byte, byte_limit;
    u32 cs = bs->prepare_cs[spi->chip_select];

    if (unlikely(!tfr->len)) {
        static int warned;

        if (!warned)
            dev_warn(&spi->dev,
                 "zero-length SPI transfer ignored\n");
        warned = 1;
        return 0;
    }

    /* set clock */
    spi_hz = tfr->speed_hz;
```

Para empezar, hay que cambiar por completo el prototipo de la función, quitar las variables locales y sustituirlas por la declaración de una variable cdiv. Al hacer el cambio del prototipo, perdemos la variable tfr con lo que no podemos dejar el fragmento de código:

```
    if (unlikely(!tfr->len)) {
        static int warned;

        if (!warned)
            dev_warn(&spi->dev,
                 "zero-length SPI transfer ignored\n");
        warned = 1;
        return 0;
```

```
    }
```

Sin embargo, este fragmento de código lo único que hace es escribir un warning que indique que se está transfiriendo un mensaje de longitud 0, y sólo lo hace la primera vez. Lo más sencillo es quitar este fragmento de código, dejando la función como sigue:

```
static unsigned long bcm2835_get_clkdiv(struct bcm2835_spi *bs, u32 spi_hz,
                    u32 *effective_speed_hz)
{
    unsigned long cdiv;

    if (spi_hz >= bs->clk_hz / 2) {
        cdiv = 2; /* clk_hz/2 is the fastest we can go */
    } else if (spi_hz) {
        /* CDIV must be a multiple of two */
        cdiv = DIV_ROUND_UP(bs->clk_hz, spi_hz);
        cdiv += (cdiv % 2);

        if (cdiv >= 65536)
            cdiv = 0; /* 0 is the slowest we can go */
    } else {
        cdiv = 0; /* 0 is the slowest we can go */
    }

    *effective_speed_hz = cdiv ? (bs->clk_hz / cdiv) : (bs->clk_hz / 65536);

    return cdiv;
}
```

Una vez hecho, podemos eliminar el fichero .rej y el .orig correspondiente:

```
$ rm drivers/spi/spi-bcm2835.c.rej drivers/spi/spi-bcm2835.c.orig
```

## 2.6 Creamos la rama 5.10.92-dovetail

Una vez parcheado el kernel creamos una rama para distinguir la versión parcheada de la original, y añadimos todos los ficheros modificados a la rama:

```
$ git checkout -b 5.10.92-dovetail
$ git add -A
$ git commit -m "dovetail patch applied"
```

## 2.7 Aplicamos el parche de Xenomai

A continuación deberemos descargarnos la versión más reciente de Xenomai, que en la fecha de redacción de este documento es la versión 3.2.1. Podemos

descargarnos este tarball o clonar el repositorio git. Vamos a optar esta segunda opción, clonaremos el repositorio en nuestro directorio de trabajo:

```
$ cd ~/rpi-kernel
$ git clone https://source.denx.de/Xenomai/xenomai.git
```

Hacemos el checkout de la rama estable 3.2.x:

```
$ git co -b v3.2.x origin/stable/v3.2.x
Branch 'v3.2.x' set up to track remote branch 'stable/v3.2.x' from 'origin'.
Switched to a new branch 'v3.2.x'
```

El proyecto utiliza autotools para la configuración y compilación, por lo que antes de trabajar con él debemos ejecutar el script bootstrap:

```
$ scripts/bootstrap
```

Ahora podemos aplicar el parche usando el script prepare-kernel.sh:

```
$ cd ../linux-xenomai
$ ../xenomai/scripts/prepare-kernel.sh --arch=arm
$ git st
On branch 5.10.92-dovetail
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   arch/arm/Makefile
        modified:   drivers/Makefile
        modified:   init/Kconfig
        modified:   kernel/Makefile

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        arch/arm/include/dovetail/
        arch/arm/xenomai/
        drivers/xenomai/
        include/asm-generic/xenomai/
        include/linux/xenomai/
        include/trace/events/cobalt-core.h
        include/trace/events/cobalt-posix.h
        include/trace/events/cobalt-rtdm.h
        include/xenomai/
        kernel/xenomai/

no changes added to commit (use "git add" and/or "git commit -a")
$ git checkout -b 5.10.92-dovetail-xenomai
$ git add -A
$ git commit "xenomai patch for cobalt applied"
```

## 2.8 Configuramos y compilamos el kernel

Ya estamos preparados para configurar y compilar el kernel.

```
$ source ../export.sh
$ make bcm2709_defconfig
$ make menuconfig
```

Seleccionamos la opción Xenomai/cobalt->Drivers y activamos los drivers del GPIO, GPIOPWM y SPI. Pueden activarse otras opciones de interés.

Después seleccionamos General Setup->Local versión y sustituimos -v7 por -v7-cobalt. Guardamos, salimos y compilamos el kernel:

```
$ make -j4 zImage modules dtbs
```

Una vez compilado, repetimos el proceso que hicimos con PREEMPT_RT para crear un tarball con los fichero a instalar en la raspberry pi. Creamos un directorio kernel-xenomai en el directorio de trabajo y lo seleccionamos como target para la instalación. Los dtbs irán en un subdirectorio boot.

```
$ mkdir ../kernel-xenomai
$ make INSTALL_MOD_PATH=../kernel-xenomai/ modules_install
$ make INSTALL_DTBS_PATH=../kernel-xenomai/boot dtbs_install
$ cp arch/arm/boot/zImage ../kernel-xenomai/boot/kernel7-xenomai.img
```

Creamos ahora el tarball cambiando el uid y el gid a 0 (root):

```
$ cd ../kernel-xenomai
$ rm lib/modules/5.10.92-v7-cobalt+/build
$ rm lib/modules/5.10.92-v7-cobalt+/source
$ fakeroot -- tar cvzf ../kernel-xenomai.tar.gz *
```

Para instalar el nuevo kernel copiamos el tarball a la raspi (sustituir la ip de la raspberry) y nos conectamos a la raspi y descomprimimos el fichero sobre el direcotrio raiz (sustituir la ip de la raspberry):

```
$ scp ../kernel-xenomai.tar.gz pi@<ip raspi>:
$ ssh pi@<ip raspi>
pi@raspberry$ sudo tar xvzf kernel-xenomai.tar.gz --keep-directory-symlink -C /
```

Finalmente, para que se seleccione este kernel en el siguiente arranque editamos el fichero /boot/config.txt y modificamos la línea que selecciona el kernel:

```
kernel=kernel7-xenomai.img
```

# 3 Librerías y utilidades de Xenomai

Las librerías xenomai son las que nos proporcionan los *skins*, y son necesarias tanto para el core Cobalt como para Mercury. Se compilan a partir del repositorio

xenomai que hemos clonado. Para hacer la compilación cruzada hacemos lo siguiente:

```
$ cd ~/rpi-kernel/xenomai
$ mkdir build-cobalt
$ cd build-cobalt
$  ../configure CFLAGS="-mtune=cortex-a53 -Wno-stringop-truncation" LDFLAGS=-mtune=cortex-a5
$ make
$ mkdir ~/rpi-kernel/xenomai-libs-cobalt
$ make DESTDIR=$(realpath ~/rpi-kernel/xenomai-libs-cobalt) install
```

Como hicimos con el kernel podemos crear un tarball para la instalación de las librerías y utilidades en la raspberry pi:

```
$ cd ~/rpi-kernel/xenomai-libs-cobalt
$ fakeroot -- tar cvzf ../xenomai-libs-cobalt.tar.gz *
```

Para instalarla procedemos como hemos hecho antes:

```
$ cd ..
$ scp xenomai-libs-cobalt.tar.gz pi@<ip raspi>:
$ ssh pi@<ip raspi>
pi@raspberry$ sudo tar xvzf xenomai-libs-cobalt.tar.gz --keep-directory-symlink -C /
```

Finalmente, vamos a añadir un parámetro de línea de comandos del kernel para que el usuario pi pueda utilizar los servicios del kernel cobalt (de lo contrario sólo root puede usarlos). Primero comprobaremos el id del usuario:

```
pi@raspberrypi:~ $ id
uid=1000(pi) gid=1000(pi) groups=1000(pi),4(adm),20(dialout),24(cdrom),27(sudo),29(audio),44
```

Con la información del id editamos el fichero /boot/cmdline.txt y añadimos a la lista de parámetros: *xenomai.allowed_group=1000*. La lista completa de los parámetros del cobalt core se puede consultar en https://source.denx.de/Xenomai/xenomai/-/wikis/Installing_Xenomai_3.

# 4 Probando la instalación

Lo primero que debemos hacer es asegurarnos de que el sistema está corriendo con el kernel cobalt (deberemos haber reiniciado):

```
pi@raspberrypi:~ $ uname -nr
raspberrypi 5.10.92-v7-cobalt+
```

Podemos también examinar la salida de dmesg, y debería salir algo parecido a esto:

```
pi@raspberrypi:~ $ dmesg | grep -i xenomai
[    0.000000] Kernel command line: coherent_pool=1M 8250.nr_uarts=1 snd_bcm2835.enable_comp
[    1.910465] [Xenomai] scheduling class idle registered.
```

```
[    1.913967] [Xenomai] scheduling class rt registered.
[    1.917654] IRQ pipeline: high-priority Xenomai stage added.
[    1.938808] [Xenomai] allowing access to group 1000
[    1.941972] [Xenomai] Cobalt v3.2.1 [LTRACE]
```

## 4.1 Tests

Muchas de las utilidades requieren ser ejecutadas por root. Para ahorrarnos poner la ruta completa vamos a añadir los directorios */usr/xenomai/bin* y */usr/xenomai/sbin* al path por defecto usado con sudo:

```
$ sudo visudo
```

Y añadimos la ruta mencionada a la variable default_path:

```
Defaults        secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/u
```

El primer tests que podemos hacer es la medida de latencias con la herramienta *latency*:

```
pi@raspberrypi:~ $ sudo latency
== Sampling period: 1000 us
== Test mode: periodic user-mode task
== All results in microseconds
warming up...
RTT|  00:00:01  (periodic user-mode task, 1000 us period, priority 99)
RTH|----lat min|----lat avg|----lat max|-overrun|---msw|---lat best|--lat worst
RTD|      3.711|      4.440|     10.809|       0|     0|      3.711|     10.809
RTD|      3.747|      4.311|      7.952|       0|     0|      3.711|     10.809
RTD|      3.852|      4.404|      7.515|       0|     0|      3.711|     10.809
RTD|      3.818|      4.423|      7.635|       0|     0|      3.711|     10.809
RTD|      2.622|      4.232|      7.770|       0|     0|      2.622|     10.809
RTD|      3.840|      4.464|      7.934|       0|     0|      2.622|     10.809
RTD|      3.808|      4.290|      6.983|       0|     0|      2.622|     10.809
RTD|      3.813|      4.349|      6.619|       0|     0|      2.622|     10.809
...
RTT|  00:07:01  (periodic user-mode task, 1000 us period, priority 99)
RTH|----lat min|----lat avg|----lat max|-overrun|---msw|---lat best|--lat worst
RTD|      3.896|      4.489|      9.540|       0|     0|     -0.681|     41.296
RTD|      3.745|      4.264|      8.813|       0|     0|     -0.681|     41.296
RTD|      3.863|      4.434|      8.195|       0|     0|     -0.681|     41.296
RTD|      3.810|      4.414|      8.939|       0|     0|     -0.681|     41.296
RTD|      3.842|      4.315|      8.358|       0|     0|     -0.681|     41.296
RTD|      3.855|      4.471|      7.660|       0|     0|     -0.681|     41.296
RTD|      3.805|      4.309|      8.453|       0|     0|     -0.681|     41.296
RTD|      2.612|      4.467|     12.857|       0|     0|     -0.681|     41.296
RTD|      3.803|      4.447|      9.675|       0|     0|     -0.681|     41.296
RTD|      3.808|      4.293|      7.500|       0|     0|     -0.681|     41.296
```

...

También podemos usar la herramienta xeno-test para valorar la latencia máxima del sistema, que hace una serie de tests y luego corre la herramienta latency con una aplicación de fondo que carga el sistema (por defecto *dohell 900*):

```
$ sudo xeno-test
pi@raspberrypi:~ $ sudo xeno-test
++ echo 0
Started child 3005: /bin/bash /usr/xenomai/bin/xeno-test-run-wrapper /usr/xenomai/bin/xeno-t
++ testdir=/usr/xenomai/bin
++ which systemctl
++ systemctl is-active --quiet systemd-timesyncd
++ timesyncd_was_running=true
++ systemctl stop systemd-timesyncd
++ /usr/xenomai/bin/smokey --run random_alloc_rounds=64 pattern_check_rounds=64
arith OK
bufp skipped (no kernel support)
cpu_affinity skipped (no kernel support)
fpu_stress OK
gdb OK
iddp skipped (no kernel support)
leaks OK
memory_coreheap OK
memory_heapmem OK
memory_tlsf OK
net_packet_dgram skipped (no kernel support)
net_packet_raw skipped (no kernel support)
net_udp skipped (no kernel support)
posix_clock OK
posix_cond OK
posix_fork OK
posix_mutex OK
posix_select OK
rtdm skipped (no kernel support)
sched_quota skipped (no kernel support)
sched_tp skipped (no kernel support)
setsched OK
sigdebug skipped (no kernel support)
timerfd OK
tsc OK
vdso_access OK
xddp skipped (no kernel support)
syscall-tests.c:1018, socket(AF_RTIPC, SOCK_DGRAM, IPCPROTO_XDDP): Address family not suppor
recvmmsg64: skipped. (no kernel support)
y2038 OK
memory_pshared OK
```

```
++ start_timesyncd
++ true
++ systemctl start systemd-timesyncd
++ timesyncd_was_running=false
++ /usr/xenomai/bin/clocktest -D -T 30 -C CLOCK_HOST_REALTIME
XNVDSO_FEAT_HOST_REALTIME not available
== Testing built-in CLOCK_HOST_REALTIME (32)
CPU      ToD offset [us] ToD drift [us/s]      warps max delta [us]
--- -------------------- ---------------- ---------- --------------
  0                  0.8           -0.001          0            0.0
  1                  1.0            0.001          0            0.0
  2                  1.4           -0.002          0            0.0
  3                  1.4            0.016          0            0.0
++ /usr/xenomai/bin/switchtest -T 30
== Testing FPU check routines...
...
++ start_load
++ echo start_load
++ false
++ check_alive /usr/xenomai/bin/latency
++ echo check_alive /usr/xenomai/bin/latency
++ wait_load
++ read rc
Started child 15812: dohell 900
Started child 15820: /usr/xenomai/bin/latency
== Sampling period: 1000 us
== Test mode: periodic user-mode task
== All results in microseconds
warming up...
RTT|  00:00:01  (periodic user-mode task, 1000 us period, priority 99)
RTH|----lat min|----lat avg|----lat max|-overrun|---msw|---lat best|--lat worst
RTD|      7.431|     18.555|     48.189|       0|     0|      7.431|     48.189
RTD|      6.860|     19.171|     39.100|       0|     0|      6.860|     48.189
RTD|      6.882|     18.866|     55.863|       0|     0|      6.860|     55.863
RTD|      6.496|     18.639|     43.737|       0|     0|      6.496|     55.863
RTD|      6.598|     19.294|     41.522|       0|     0|      6.496|     55.863
RTD|      7.071|     19.278|     41.830|       0|     0|      6.496|     55.863
RTD|      6.813|     19.003|     44.208|       0|     0|      6.496|     55.863
RTD|      7.129|     18.809|     45.973|       0|     0|      6.496|     55.863
RTD|      6.571|     18.044|     48.458|       0|     0|      6.496|     55.863
RTD|      7.109|     18.189|     50.539|       0|     0|      6.496|     55.863
RTD|      7.098|     17.502|     45.169|       0|     0|      6.496|     55.863
RTD|      6.912|     16.984|     42.326|       0|     0|      6.496|     55.863
RTD|      7.427|     19.228|     44.753|       0|     0|      6.496|     55.863
RTD|      6.576|     19.732|     47.160|       0|     0|      6.496|     55.863
RTD|      7.114|     18.661|     42.139|       0|     0|      6.496|     55.863
```

```
...

RTH|----lat min|----lat avg|----lat max|-overrun|---msw|---lat best|--lat worst
RTD|      6.990|     18.403|     40.851|       0|     0|      5.279|     62.183
child 15812 returned: killed by signal 9
Load script terminated, terminating checked scripts
---|-----------|-----------|-----------|--------|------|-------------------------
RTS|      5.279|     18.803|     62.183|       0|     0|    00:15:04/00:15:04
child 15820 returned: exited with status 0
pipe_in: /tmp/xeno-test-in-3005
+ start_timesyncd
+ false
child 3005 returned: exited with status 0
```

La latencia observada arriba sólo es el peor de los casos en los 15 minutos de la ejecución por defecto, no debe tomarse como la latencia máxima posible, hay que hacer pruebas mucho más largas.

También se puede usar la herramienta autotune para calibrar el timer del core Cobalt:

```
pi@raspberrypi:~ $ sudo autotune
== auto-tuning started, period=1000000 ns (may take a while)
irq gravity... 1500 ns
kernel gravity... 3000 ns
user gravity... 5500 ns
== auto-tuning completed after 30s
```